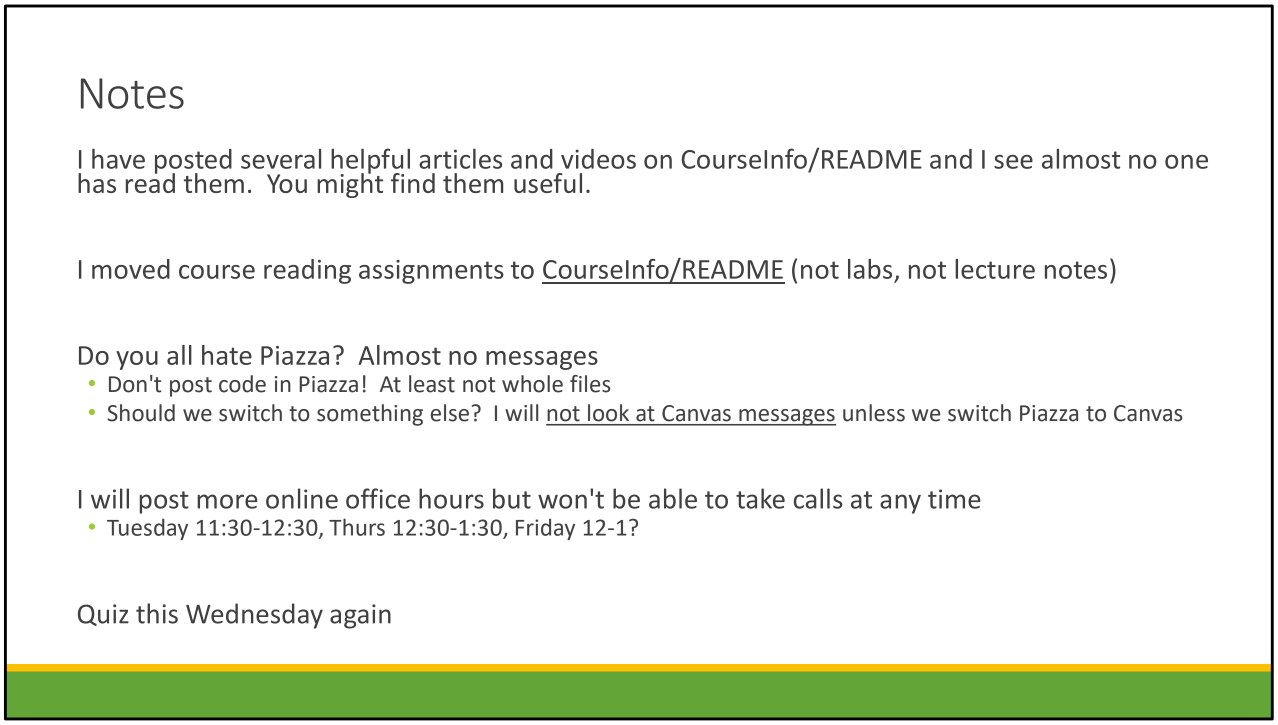
CS112 –
Java
Programming

Fall 2022

More Object Oriented Programming



Notes

I have posted several helpful articles and videos on CourseInfo/README and I see almost no one has read them. You might find them useful.

I moved course reading assignments to CourseInfo/README (not labs, not lecture notes)

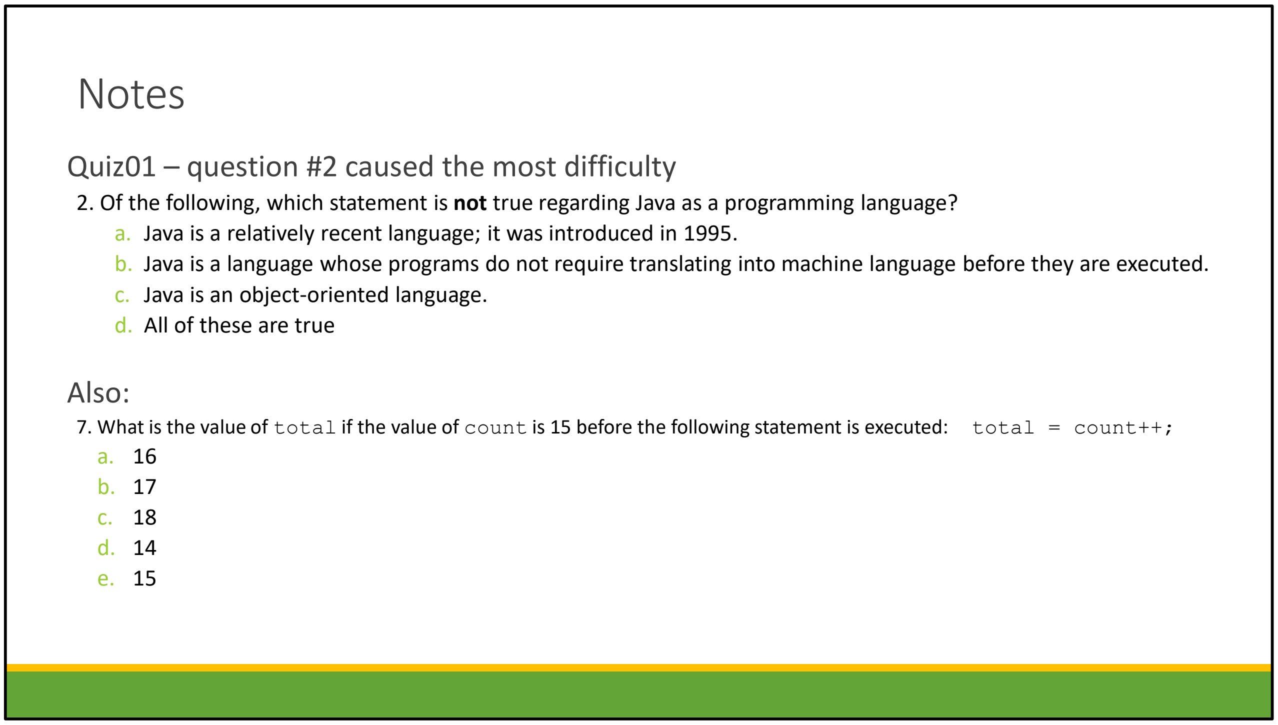
Do you all hate Piazza? Almost no messages

- Don't post code in Piazza! At least not whole files
- Should we switch to something else? I will not look at Canvas messages unless we switch Piazza to Canvas

I will post more online office hours but won't be able to take calls at any time

- Tuesday 11:30-12:30, Thurs 12:30-1:30, Friday 12-1?

Quiz this Wednesday again



Notes

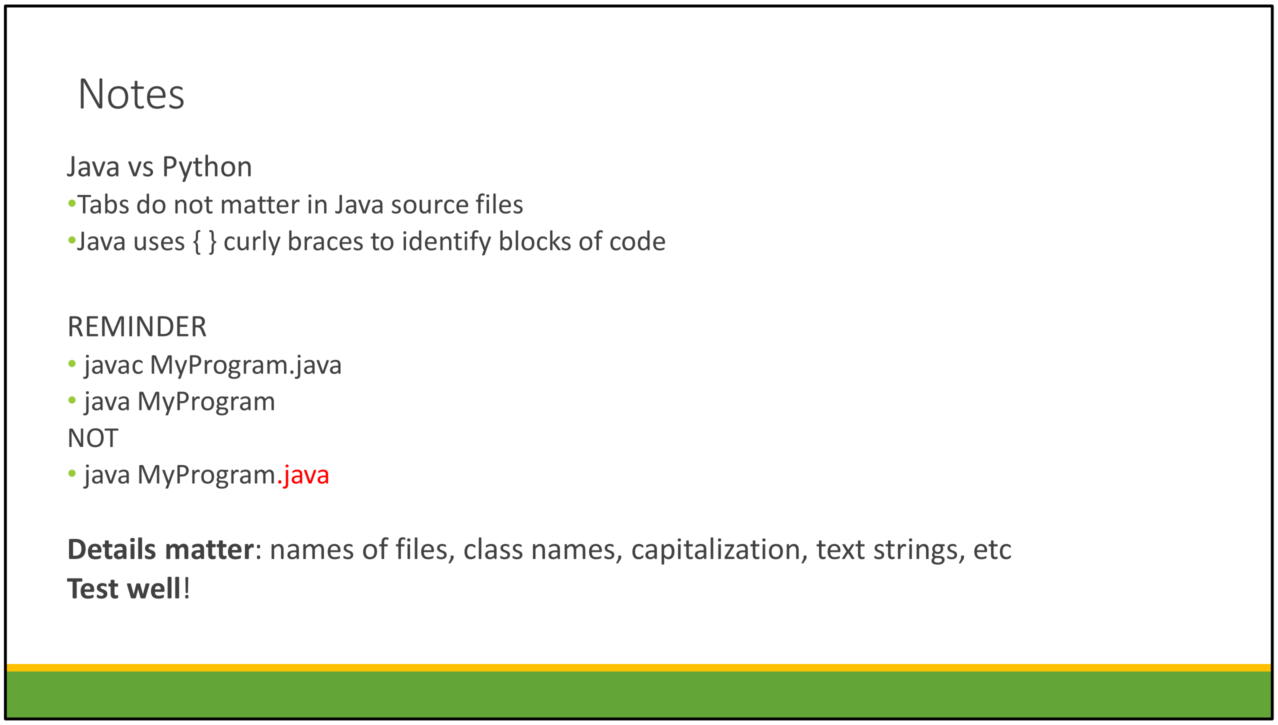
Quiz01 – question #2 caused the most difficulty

2. Of the following, which statement is **not** true regarding Java as a programming language?
   a. Java is a relatively recent language; it was introduced in 1995.
   b. Java is a language whose programs do not require translating into machine language before they are executed.
   c. Java is an object-oriented language.
   d. All of these are true

Also:

7. What is the value of `total` if the value of `count` is 15 before the following statement is executed: `total = count++;`
   a. 16
   b. 17
   c. 18
   d. 14
   e. 15

Order of operations: (Prefix operator)P(cast)MDAS(Assignment)(Postfix operator)



Notes

Java vs Python

- Tabs do not matter in Java source files
- Java uses { } curly braces to identify blocks of code

REMINDER

- javac MyProgram.java
- java MyProgram

NOT

- java MyProgram.java

Details matter: names of files, class names, capitalization, text strings, etc

Test well!

This class is not to teach you Java--that would take 3 weeks.
It is to teach you to be a high quality programmer, able to work through difficult assignments.

Last time…

We discussed

- Procedural programming and Object Oriented Programming
- Ideas for good class design
- Syntax for classes and methods

Encapsulation

We can take one of two views of an object:

- internal - the details of the variables and methods of the class that defines it
- external - the services that an object provides and how the object interacts with the rest of the system

From the external view, an object is an *encapsulated* entity, providing a set of specific services

These services define the *interface* to the object

Encapsulation

One object (called the *client*) may use another object for the services it provides

The client of an object may request its services (call its methods), but it should not have to be aware of how those services are accomplished

Any changes to the object's state (its variables) should be made by that object's methods

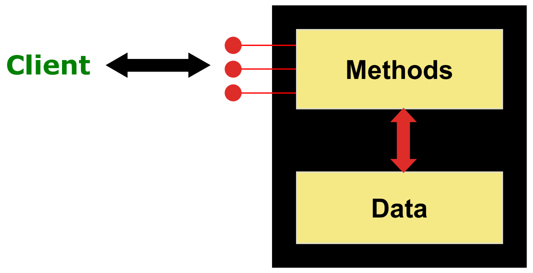
We should make it difficult, if not impossible, for a client to access an object’s variables directly

That is, an object should be *self-governing*

Encapsulation

An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client

The client invokes the interface methods and they manage the instance data



Visibility Modifiers

In Java, we accomplish encapsulation through the appropriate use of *visibility modifiers*

A *modifier* is a Java reserved word that specifies particular characteristics of a method or data

We've used the `final` modifier to define constants

Java has three visibility modifiers: `public`, `protected`, and `private`

The `protected` modifier involves inheritance, which we will discuss later

Visibility Modifiers

Members (variables or methods) of a class that are declared with *public visibility* can be referenced anywhere

Members of a class that are declared with *private visibility* can be referenced only within that class

Members declared without a visibility modifier have *default visibility* and can be referenced by any class in the same `package` (which we will discuss later)

Visibility Modifiers

Public variables violate encapsulation because they allow the client to modify the values directly

Therefore instance variables should not be declared with public visibility

It is acceptable to give a constant public visibility, which allows it to be used outside of the class

Public constants do not violate encapsulation because, although the client can access it, its value cannot be changed

Visibility Modifiers

Methods that provide the object's services are declared with public visibility so that they can be invoked by clients

Public methods are also called *service methods*

A method created simply to assist a service method is called a *support method*

Since a support method is not intended to be called by a client, it should not be declared with public visibility

Visibility Modifiers

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**

Accessors and Mutators

Because instance data is private, a class usually provides services to access and modify data values

An *accessor method* returns the current value of a variable

A *mutator method* changes the value of a variable

The names of accessor and mutator methods take the form `getX` and `setX`, respectively, where `X` is the name of the value

They are sometimes called “getters” and “setters”

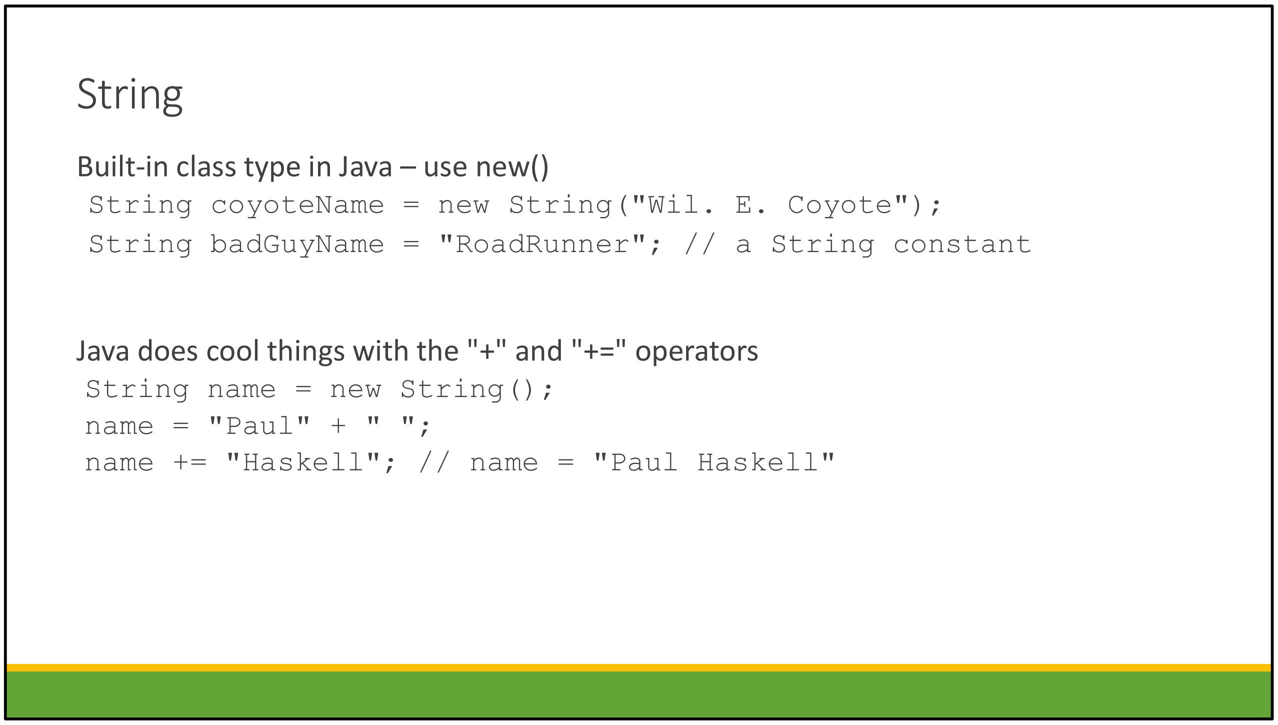
Mutator Restrictions

The use of mutators gives the class designer the ability to restrict a client's options to modify an object's state

A mutator is often designed so that the values of variables can be set only within particular limits

For example, width and height of an image must be >0.

class String



String

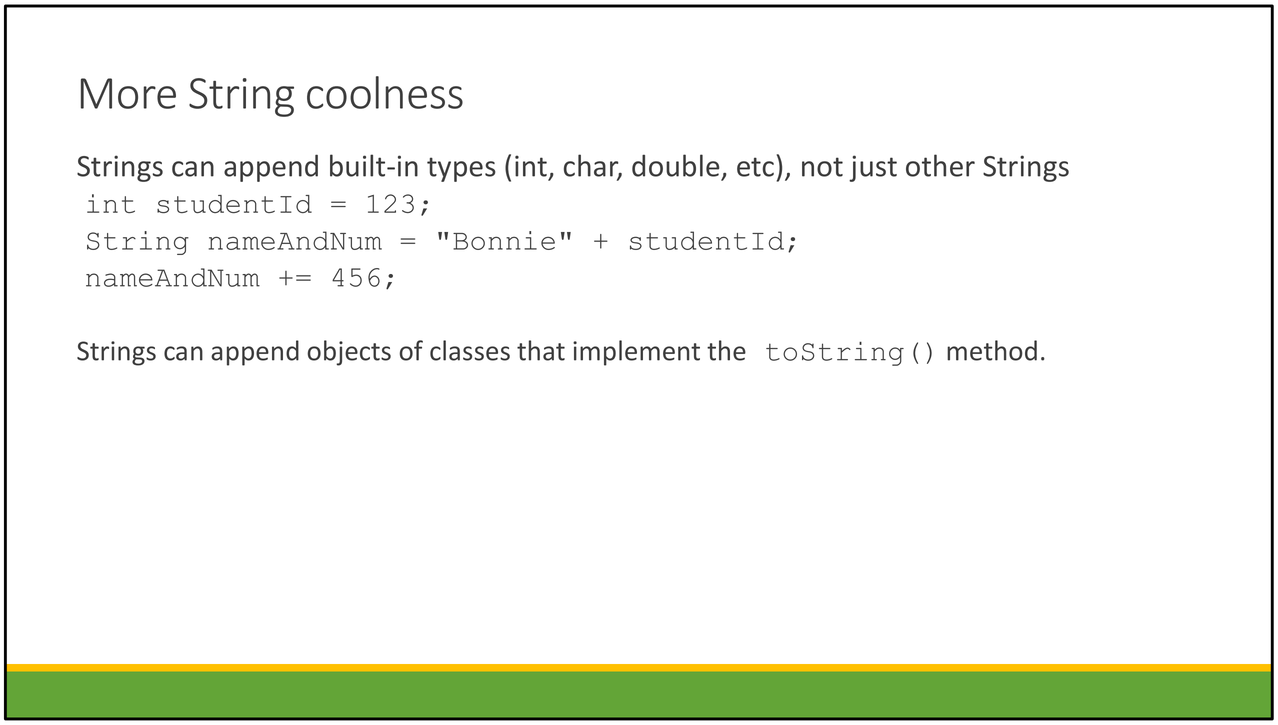
Built-in class type in Java – use new()

```
String coyoteName = new String("Wil. E. Coyote");
String badGuyName = "RoadRunner"; // a String constant
```

Java does cool things with the "+" and "+=" operators

```
String name = new String();
name = "Paul" + " ";
name += "Haskell"; // name = "Paul Haskell"
```

String is a class, but it is a SPECIAL BUILT-IN class, with some special capabilities, like initialization directly from a character constant.



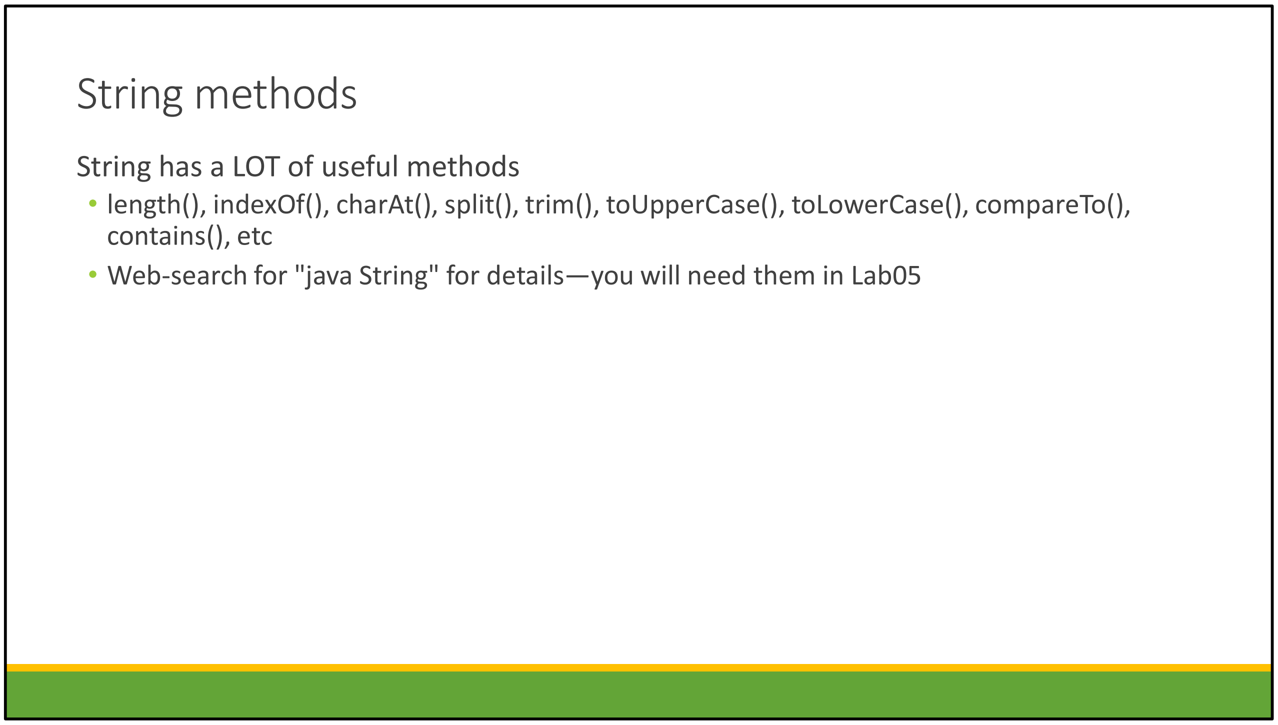
More String coolness

Strings can append built-in types (int, char, double, etc), not just other Strings

```
int studentId = 123;
String nameAndNum = "Bonnie" + studentId;
nameAndNum += 456;
```

Strings can append objects of classes that implement the `toString()` method.

Another special behavior for String. Our classes cannot use +=



String methods

String has a LOT of useful methods

- length(), indexOf(), charAt(), split(), trim(), toUpperCase(), toLowerCase(), compareTo(), contains(), etc
- Web-search for "java String" for details—you will need them in Lab05

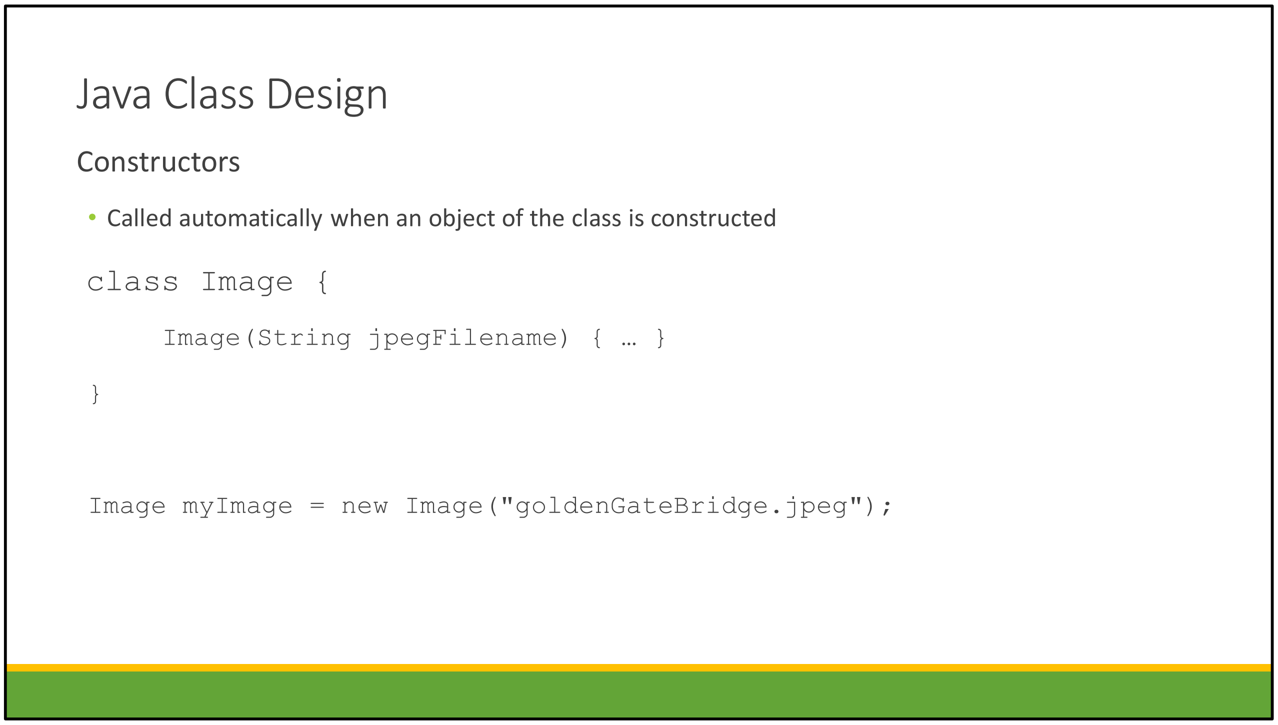
TRY THESE OUT IN ECLIPSE!!

Special Characters

"Escape sequences" let you represent some special-case characters in String constants:

- Newline = '\n'
- Tab = '\t'
- Double-quote = '\"'
- Backslash = '\\'

Class Constructors



Java Class Design

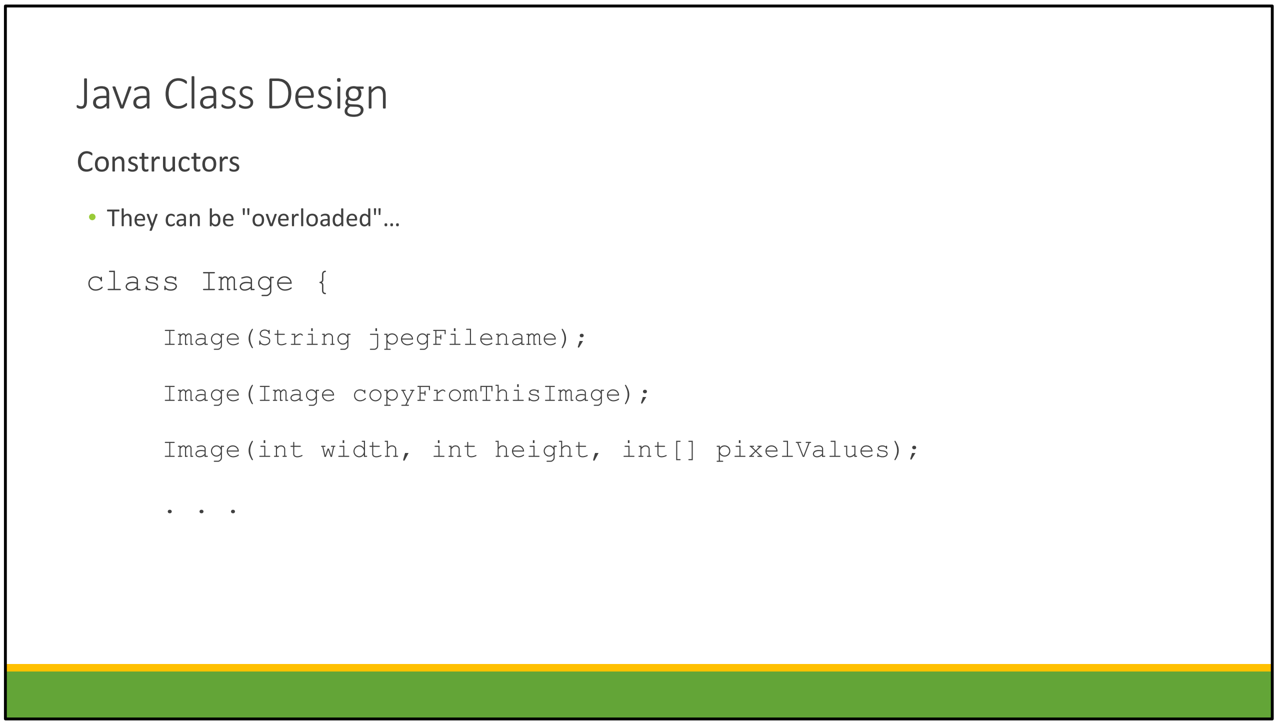
Constructors

- Called automatically when an object of the class is constructed

```
class Image {
     Image(String jpegFilename) { … }
}


Image myImage = new Image("goldenGateBridge.jpeg");
```

Remember __init__() in Python



Java Class Design

Constructors

- They can be "overloaded"...

```
class Image {

     Image(String jpegFilename);

     Image(Image copyFromThisImage);

     Image(int width, int height, int[] pixelValues);

     . . .
```

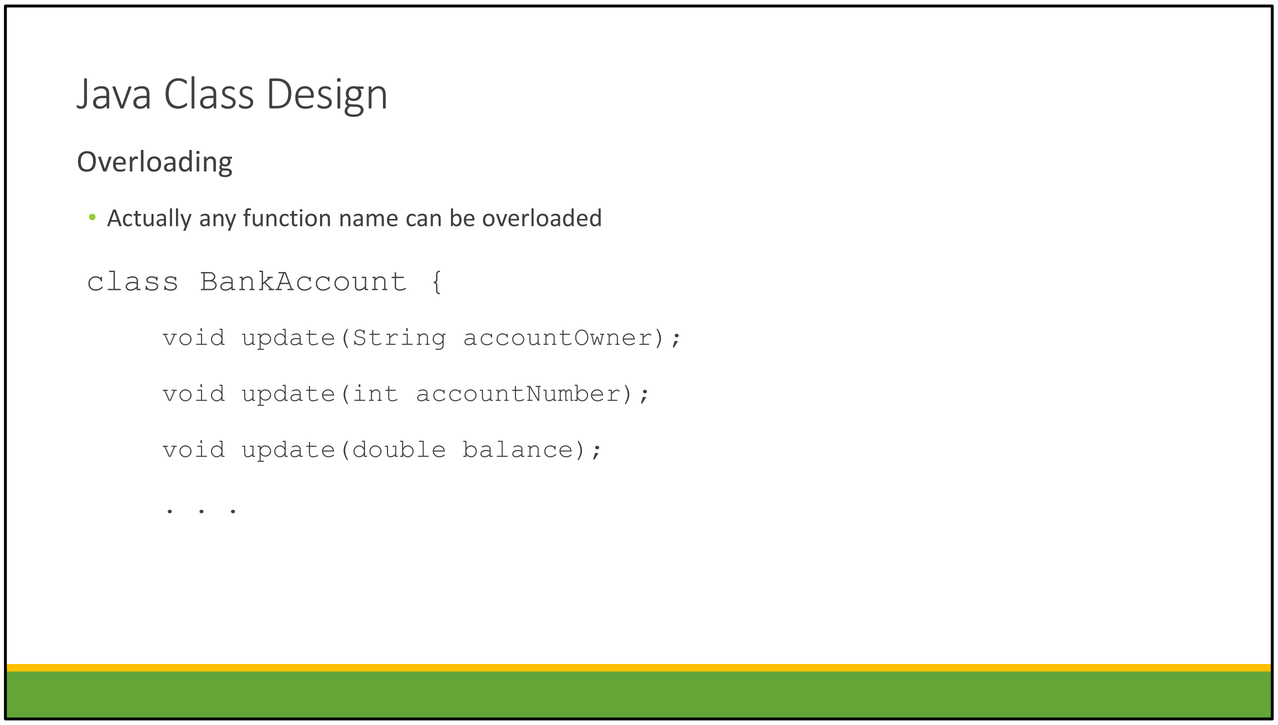
OVERLOADING: multiple different versions of the same function. Different arguments
Cannot differ by return values, only by arguments. All must have same return types! Each has own code

Java Class Design

One constructor can call another, using `this` keyword.

`this` lets source code refer to the current object.

```
 class Image {
      Image(String jpegFilename) {
            ReadJpegFile(jpegFilename); // reads width, height, pixels
            this(imageWidth, imageHeight, pixels); // creates Image
            . . .
      }
```



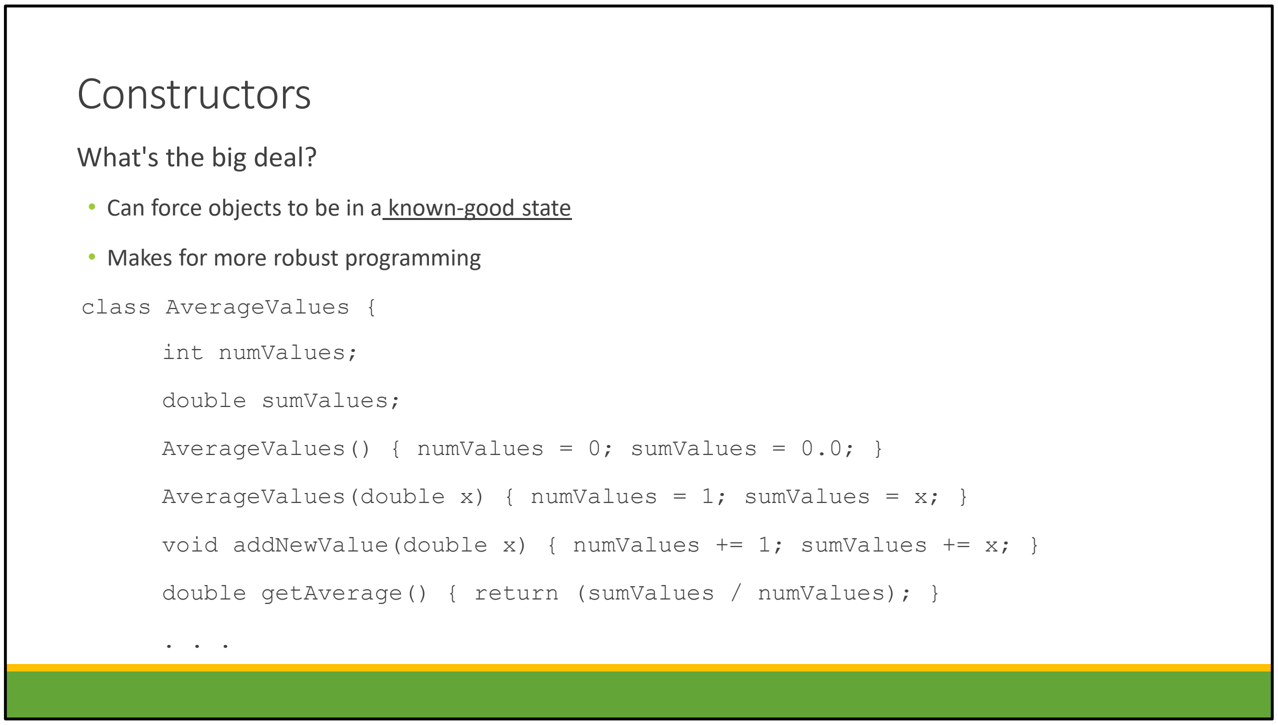
Java Class Design

Overloading

- Actually any function name can be overloaded

```
class BankAccount {
     void update(String accountOwner);
     void update(int accountNumber);
     void update(double balance);
     . . .
```

Those last two are very risky: it is not easy to be careful whether we pass an int or double to the function
CAN WE OVERLOAD BY RETURN TYPE?



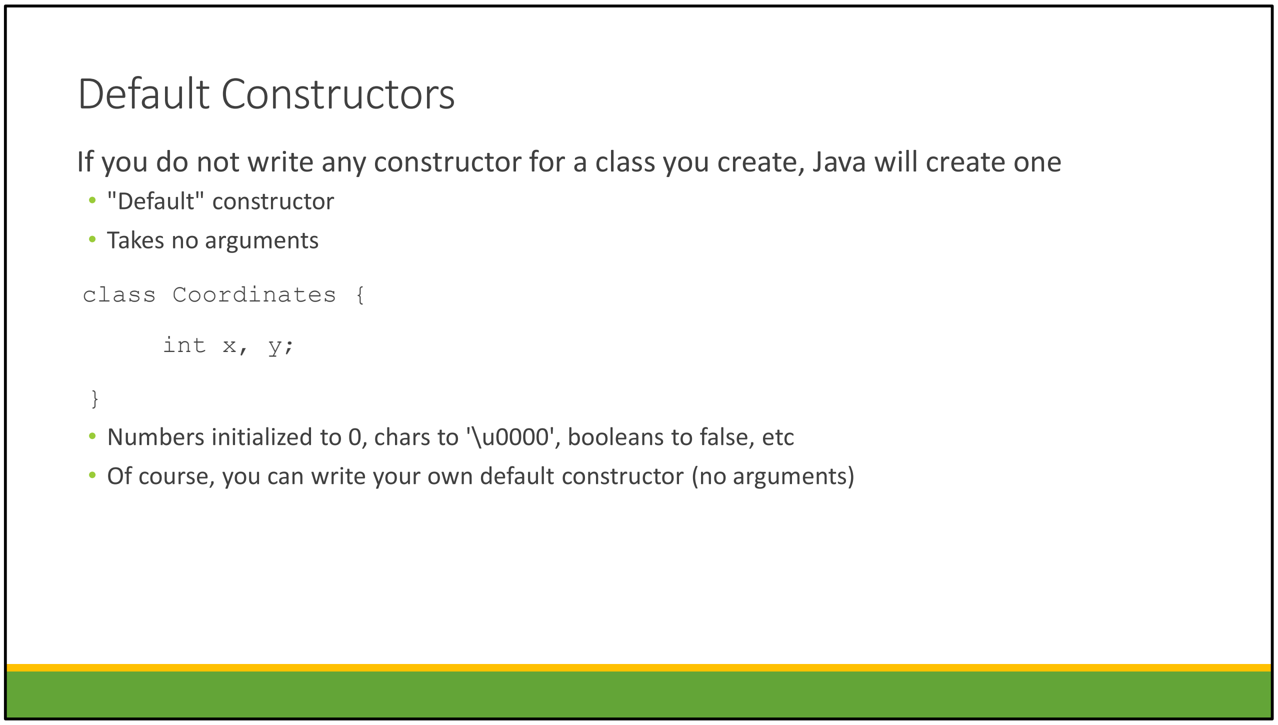
Constructors

What's the big deal?

- Can force objects to be in a known-good state
- Makes for more robust programming

```
class AverageValues {
      int numValues;
      double sumValues;
      AverageValues() { numValues = 0; sumValues = 0.0; }
      AverageValues(double x) { numValues = 1; sumValues = x; }
      void addNewValue(double x) { numValues += 1; sumValues += x; }
      double getAverage() { return (sumValues / numValues); }
      . . .
```

Without the constructor, we don't know if our two member variables have been initialized properly.
In our class Sphere, would be better to set diameter from constructor, rather than setDiameter method: forces objects to be initialized!



Default Constructors

If you do not write any constructor for a class you create, Java will create one

- "Default" constructor
- Takes no arguments

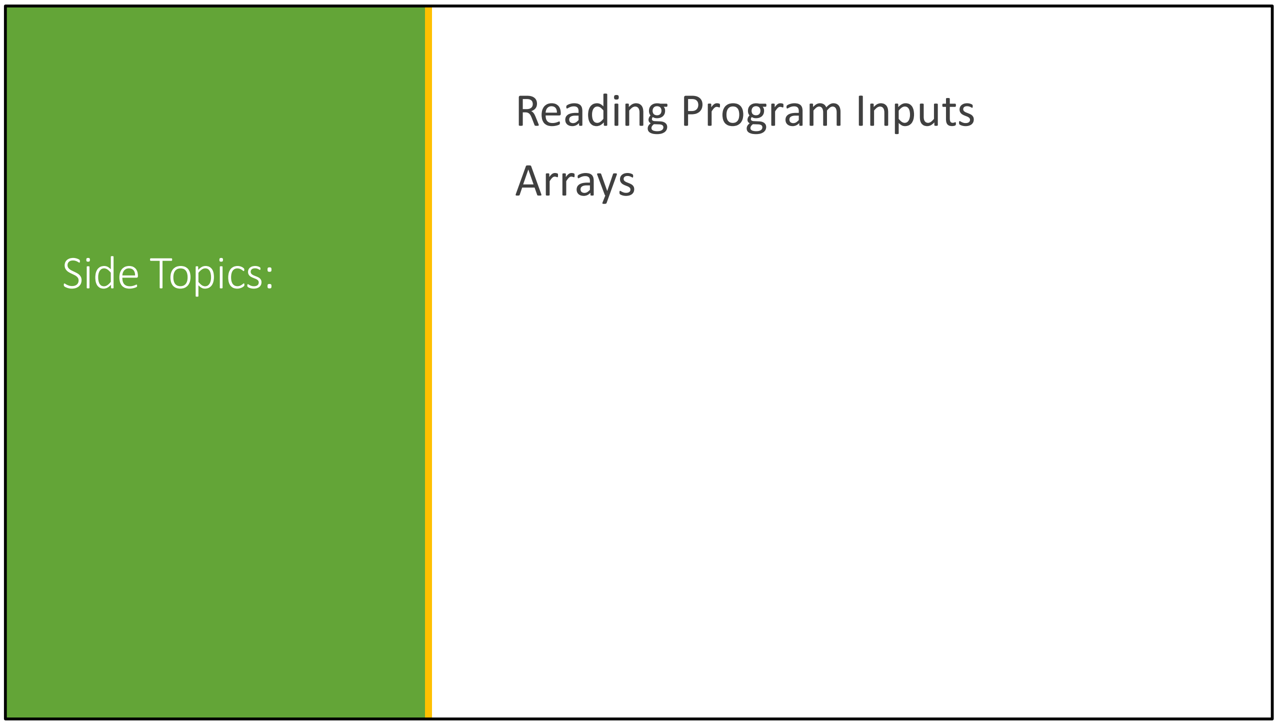
```
class Coordinates {

      int x, y;

 }
```

- Numbers initialized to 0, chars to '\u0000', booleans to false, etc
- Of course, you can write your own default constructor (no arguments)

Java creates default constructor if you don't write any constructors.
If you write any, Java will not create one itself. You can of course create a constructor w no args.



Side Topics:

Reading Program Inputs

Arrays

Inputs can come from several places: keyboard, files, internet connections, etc. We start with COMMAND LINE ARGUMENTS

Arrays

`String[]` is an array

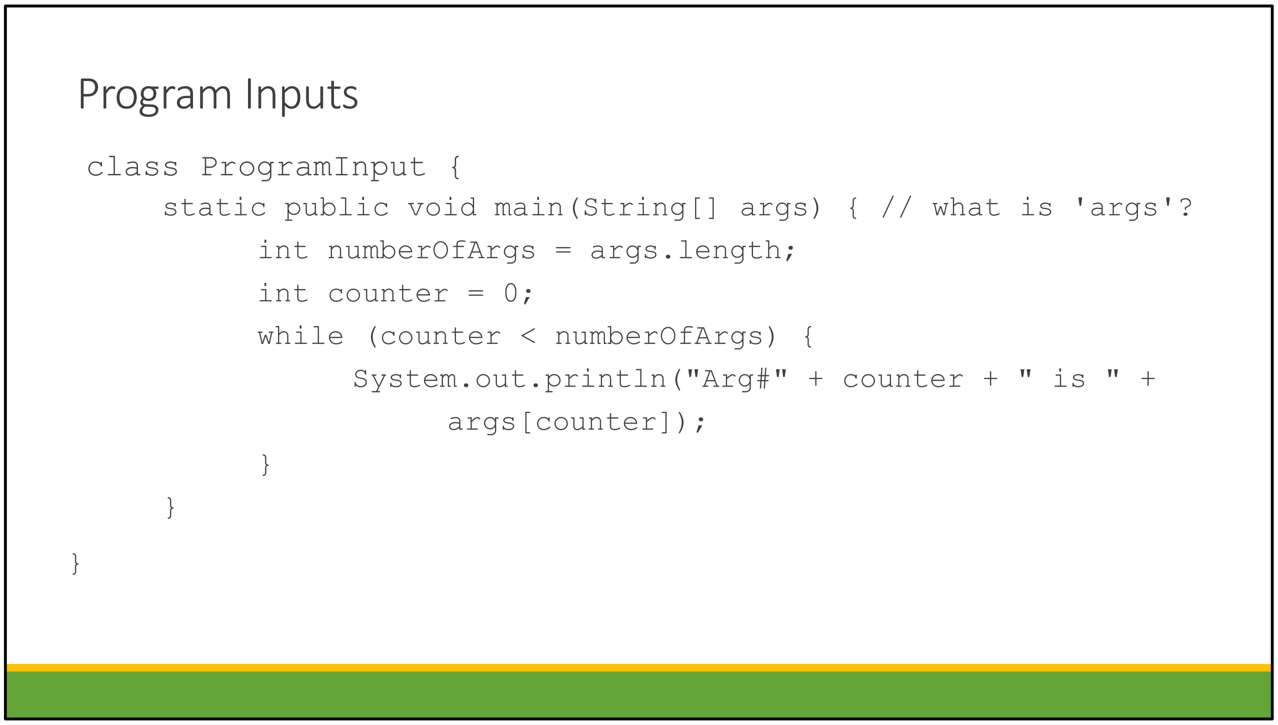
- Array = sequence of elements all with the given data type
- Can create with any data type
- Arrays have a `length` member variable that shows their length (not member function as with class String)

How to create the array?

```
  int NumberOfElements = ???;
  String arrayOfStrings[] = new String[NumberOfElements];
```

The elements in the array are not initialized. They must be before they are used…

```
int i = 0;
while (i < arrayOfStrings.length) {
  arrayOfStrings[i] = new String();
  i += 1;
 }
```

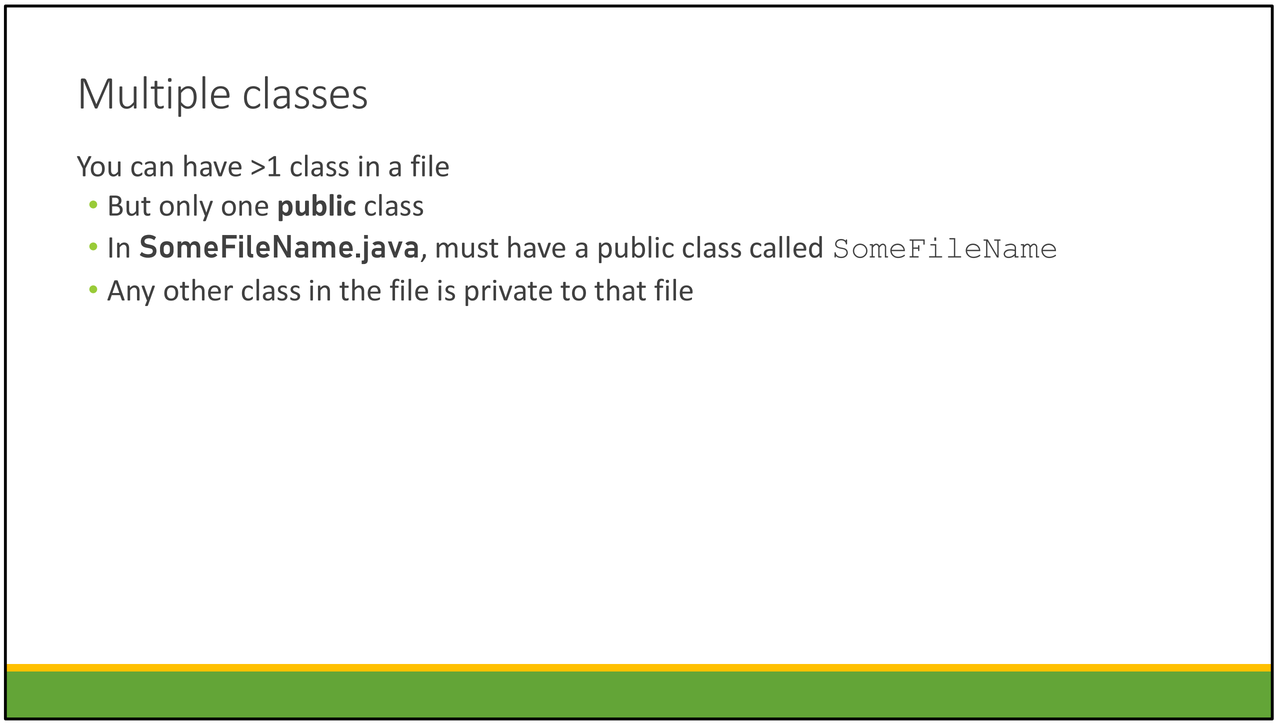


Program Inputs

```
 class ProgramInput {
     static public void main(String[] args) { // what is 'args'?
          int numberOfArgs = args.length;
          int counter = 0;
          while (counter < numberOfArgs) {
               System.out.println("Arg#" + counter + " is " +
                    args[counter]);
          }
     }
}
```

Arguments on the program command-line!

One more thing



Multiple classes

You can have >1 class in a file

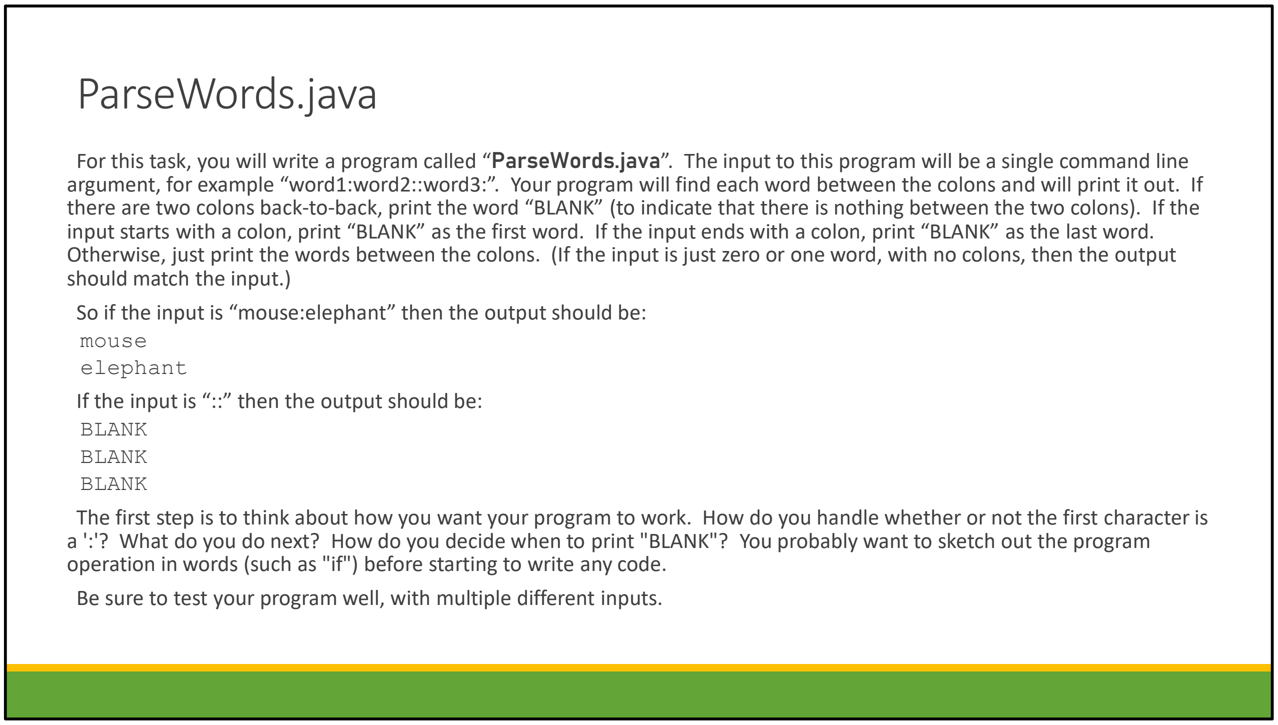
- But only one **public** class
- In **SomeFileName.java**, must have a public class called `SomeFileName`
- Any other class in the file is private to that file

A Public class can be used by other classes in other files
The public class is the one with main()

Lab05

Removed hardest problem……….

so we can do together in class!



ParseWords.java

For this task, you will write a program called “**ParseWords.java**”. The input to this program will be a single command line argument, for example “word1:word2::word3:”. Your program will find each word between the colons and will print it out. If there are two colons back-to-back, print the word “BLANK” (to indicate that there is nothing between the two colons). If the input starts with a colon, print “BLANK” as the first word. If the input ends with a colon, print “BLANK” as the last word. Otherwise, just print the words between the colons. (If the input is just zero or one word, with no colons, then the output should match the input.)

So if the input is “mouse:elephant” then the output should be:

```
mouse
elephant
```

If the input is “::” then the output should be:

```
BLANK
BLANK
BLANK
```

The first step is to think about how you want your program to work. How do you handle whether or not the first character is a ':'? What do you do next? How do you decide when to print "BLANK"? You probably want to sketch out the program operation in words (such as "if") before starting to write any code.

Be sure to test your program well, with multiple different inputs.

Blackboard design first!
Want a PrevWasColon Boolean variable. Tells us whether to print BLANK
Then coding!
Then testing!

Lab05

Let's take a look…